

MOOLAP: Towards Multi-Objective OLAP *

Shyam Antony, Ping Wu, Divyakant Agrawal, Amr El Abbadi

University of California, Santa Barbara
California, USA

{shyam, pingwu, agrawal, amr}@cs.ucsb.edu

Abstract—Aggregation is among the core functionalities of OLAP systems. Frequently, such queries are issued in decision support systems to identify interesting groups of data. When more than one aggregation function is involved and the notion of interest is not clearly defined, skyline queries provide a robust mechanism to capture the potentially interesting points where (i) users do not need to specify a ranking function and (ii) the result is independent of the dimension scales. To provide better exploration functionalities in OLAP systems, we propose to use skyline queries over aggregated data to identify the *most interesting groups*. Since aggregation functions have to be ad-hoc to cover a wide variety of user interests, the skyline over the aggregates has to be computed on the fly. Hence any algorithm to compute such a skyline must be fast and be able to progressively produce the result set with potential skyline groups being produced as early as possible. We explore a family of algorithms which try to consume only as many data records as are necessary to compute the skyline and design an optimal algorithm. We further refine the algorithm by taking into account systems issues such as disk behavior which are often ignored but have strong impact on real system performance. Experimental results validate the performance and progressive benefits of our algorithm.

I. INTRODUCTION

Multi-Objective optimization (MOO) has been an important subject of decision making for many decades. It deals with problems that seek to simultaneously optimize multiple objectives. For such problems, often there does not exist one single optimal solution in the traditional sense. Instead, a number of alternative solutions may coexist involving various trade-offs in different objectives.

In recent years, the skyline query operator has attracted considerable amount of attention. Such interest is mainly driven by the fundamental need for MOO in decision making and data analysis scenarios. In fact, skyline queries can be seen as an instance of MOO where each tuple in the database is considered a feasible solution in a discrete solution space and attributes of interest are considered as the optimization goals.

Existing research on skyline queries has been mainly restricted to the problem of selecting “interesting” objects/tuples from OLTP databases. However, current enterprise decision making systems benefit more from OLAP than OLTP. Due to the often large and complex data spaces in OLAP, analysts need the system to prioritize the cube cells and highlight interesting subspaces. In many scenarios, users often do not know how to quantify the trade-offs between different

goals and just wish to quickly grasp what can be potentially interesting. Furthermore, different user objectives functions can be conflicting in nature. This essentially calls for a new functionality which skyline operations can readily offer.

In order to introduce the skyline operation into existing OLAP systems, the main challenge is to efficiently compute *skyline over aggregation*, which although useful, has not been studied in previous literature. The work closest to our work was done in the top-k context by Li et al. [1]. Details omitted in this brief paper are available in the full version [2].

A. Query Model

Consider an OLAP fact table FT with p dimensional attributes D_1, D_2, \dots, D_p and k measure attributes M_1, M_2, \dots, M_k . A skyline aggregation query consists of two basic components: group-by predicates and objective functions. Each objective function OBJ_i maps a group of tuples to a scalar value and can be defined by users in an ad-hoc manner. Such a query can be formulated as follows:

```
SELECT  $D_1, \dots, D_p, OBJ_1, \dots, OBJ_d$ 
```

```
FROM  $FT$ 
```

```
GROUP BY  $D_1, \dots, D_p$ 
```

```
SKYLINE ON  $OBJ_1, OBJ_2, \dots, OBJ_d$ 
```

where OBJ_1, \dots, OBJ_d are d objective functions defined over the measure attributes $\{M_1, M_2, \dots, M_k\}$. A skyline aggregation query first partitions the tuples in FT into different groups based on the *grouping attributes*, and for each the resulting group, d values are computed based on the d objective functions $\{OBJ_1, OBJ_2, \dots, OBJ_d\}$ defined in the **SKYLINE ON** clause. A group g_1 is said to be dominated by another group g_2 if for every objective OBJ_i , $OBJ_i(g_2) \leq OBJ_i(g_1)$ and for at least one objective OBJ_j we have $OBJ_j(g_2) < OBJ_j(g_1)$. The skyline groups are those groups that are not dominated by any other group.

An objective is determined by two components, an aggregate function A and a feature expression F . Among these two components, feature expressions tend to be user defined while aggregate functions tend to be pre-defined system functions. We allow arbitrary monotonic feature expressions. The solutions we develop are correct for monotone aggregate functions and efficient for self-maintainable aggregate functions.

II. SOLUTION FRAMEWORK

A skyline aggregation query consists of a grouping component and an objective component. The baseline reference solution first aggregates all groups. Since the grouping part

*This work is supported in part by NSF grant IIS 0223022.

is pre-computed, the aggregation phase can be accomplished with a single pass over the data. The aggregated groups are then input into a skyline computation algorithm. The prime target for optimization is the aggregation phase since its input is much larger than the input to the skyline computation phase.

Multi-Dimensional Bounds: Bounds of a d -dimensional point are essentially points which are either identical to the point or located in the region that dominates the point. The *tightest bound* is that which is identical to the actual position of the group in the objective space. The main intuition behind our solution is to compute the bounds of groups using read tuples and use the bounds to prune out non-skyline groups as early as possible.

Bounding Strategy: How do we determine a bound for a group before any tuples of that group is read? How do we tighten that bound whenever a tuple (or set of tuples) is read? And finally, in what order should we read the different tuples of a group? We call an answer to this three part question as a *bounding strategy*.

We consider the family of all algorithms that use the same bounding strategy and only use the information provided by the bounds in determining the skyline groups. We term this family of algorithms as the *lower bounding skyline groups algorithms (LBSGA)*. Next we state some properties of algorithms in the LBSGA family and identify one which is optimal in terms of the total number of tuples read.

Property 1: Let S denote the set of skyline groups and let g be a given non skyline group. Then the minimum number of tuples to be read from g before g is discarded by any algorithm in the LBSGA family is the minimum number of tuples needed by the bounding strategy followed to produce a bound of g that is dominated by at least one group in S .

Property 2: Let g_c be any group with unread tuples in the current skyline set S_c . Then no correct algorithm in LBSGA can terminate without reading more tuples from g_c . Furthermore, if for every group in S_c all tuples have been read, then $S_c = S$.

A. MOOLAP Algorithm

We exploit the fact that the bound with the minimum score with respect to some monotone function, say the L_1 norm, always belongs to the current skyline. We term the monotone function the *group ranking function*. We arrange the bounds in an external memory priority queue on the basis of their *group ranking*. The final skyline set S is initially empty. At each step we retrieve the bound with minimum group ranking function score from the priority queue and check if it is dominated by any group in the final skyline set S . If it is dominated it is immediately discarded. Otherwise we use the bounding strategy to read more tuples and tighten its bound. The group bound is then again checked for dominance and if it is dominated by some group in S , it is discarded. If the group bound is fully materialized, i.e., it does not have another unread tuple it is added to the final skyline set S . Otherwise it is reinserted into the priority queue Q on the basis of its adjusted group ranking function score.

Theorem 1: If the MOOLAP algorithm is run to completion, then every skyline group is fully materialized. Furthermore, the skyline groups are materialized in non-decreasing order of their group ranking function score.

Theorem 2: The MOOLAP algorithm is an optimal algorithm in the LBSGA family defined by a given bounding strategy where the cost is the number of tuples read.

B. Sequential Index Bounding Strategy

The basic idea is that if we precompute and store succinct statistics about the tuples of a group, we can come up with tight bounds. Our first bounding strategy, which we call the *Index Bounding Strategy (IBS)*, works as follows. In a pre-processing step partition the tuples of a group into different pages. Precompute and store meta-information necessary to lower bound all the tuples in each page. Then the initial bound for the tuples is obtained by reading just the meta information. Subsequently whenever a group makes it to the head of the priority queue in the MOOLAP algorithm, its bound is refined by reading one unread page and replacing the estimate provided by the meta-information for that page with the actual information.

We divide the pages into index pages and data pages. Index pages hold the meta-data while data-pages store the corresponding tuples. For each data page, an entry in the index page stores the minimum value of each measure and the number of tuples in that data page. Since we assume that the feature expressions and aggregation functions are monotonic this information is sufficient to obtain a bound.

So far in the discussion, we assumed that savings in terms of number of tuples consumed is necessary and sufficient for savings in time and hence optimized only the number of tuples. But empirically, this hypothesis is not true and other factors need to be taken into account. The factor that is most pertinent is disk behavior. Sequential reading of data pages is orders of magnitude faster than the a read which involves a random seek. Now, we develop a bounding strategy called the sequential index bounding strategy (SIBS) that takes into account the disk access time.

SIBS maintains a set of potential skyline groups in addition to the set of final skyline groups. Initially both sets are empty. In a pre-processing step, SIBS builds index pages and the index pages store the minimum measures for each data page. All the index pages are read in the beginning to compute the initial bound of all groups and the priority queue of the MOOLAP algorithm is organized as before. Group g is popped from the head of the priority queue. If g is dominated by some group in either the potential skyline set or the final skyline set, g is discarded. If it is not dominated then instead of inserting the group back into the priority queue, the sequentially next page of g is read and the process is repeated. When all the pages of g are read, g is inserted back into the priority queue and also into the potential skyline set if the potential skyline set has size less than its fixed capacity. When g makes it to the head of the priority queue again and if it is still not dominated by either the members of the potential

skyline set or the final skyline set it is removed from the potential skyline set and added to the final skyline set. The next section shows that SIBS is significantly faster than full aggregation.

III. EXPERIMENTS

We present some representative experimental results that demonstrate the performance benefits of the MOOLAP algorithm (with SIBS bounding strategy) when compared with full aggregation followed by skyline computation. Different polynomials of order 5 was used as the default feature expressions and sum is used as the aggregation function. 4 measures and 4 objectives were used as default. The number of tuples per group is normally distributed with a standard deviation of 0.2. The experiments were conducted on a dual pentium-4 processor machine with 1GB of RAM, a 500GB IDE hard disk and running a 2.6.17 linux kernel. Databases of sizes ranging from 1GB to 100GB were used. Due to severe restriction of space, only anti-correlated measures and objectives (the worst cases) are shown. Results are consistent in other distributions as well. The y -axes are in logarithmic scale.

1) *Scalability of MOOLAP*: Figure 1 examines the scalability of the MOOLAP algorithm in terms of the number of groups and tuples respectively. By varying both the average group size and the group number from 1K to 100K, we have covered all sizes of databases up to 100GB. In all cases, MOOLAP consistently outperforms the baseline method by orders of magnitude. For example, on the 100GB databases, MOOLAP completes within 3 minutes whereas the baseline method takes several hours.

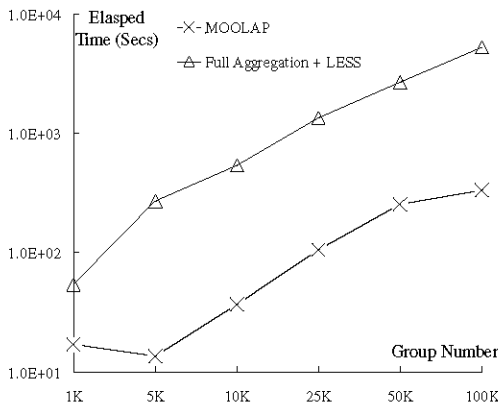


Fig. 1. Effect of Number of Groups

2) *Impact of Dimensionality*: Figure 2 shows the impact of the number of measures on the performance of MOOLAP in terms of elapsed time. When the number of measures is less than 5, MOOLAP is order of magnitude faster than the baseline. For higher number of measures, MOOLAP is still around 50-60 percent faster. The impact of the number of objectives on the performance of MOOLAP is also very similar.

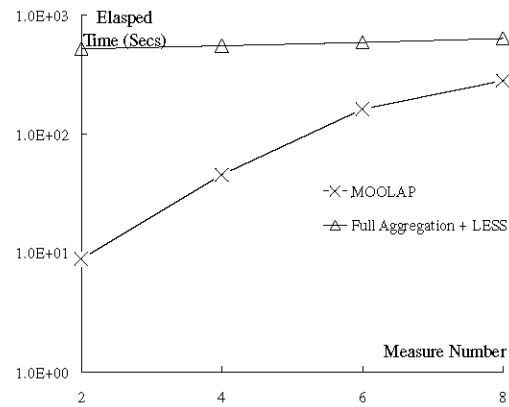


Fig. 2. Effect of Number of Measures

3) *Study of Anti-correlated Objectives*: In this experiment, we study the performance of MOOLAP on data sets that are *anti-correlated in the objective space*. As it can be seen from the curves in Figure 3, MOOLAP also suffers from the “curse of anti-correlation” which afflicts the skyline operator. Nevertheless, MOOLAP still outperforms the baseline algorithm by a wide margin (around 50%).

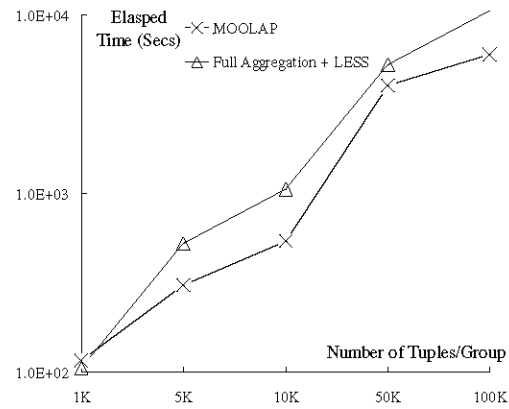


Fig. 3. Effect of Objective Anti-Correlation

4) *Miscellaneous experimental Observations*: We summarize a few other experimental observations. Details are omitted due to lack of space but are available in the full version of this paper [2]. As the rate of increase of the feature expression increases performance degrades but stabilizes beyond a certain point. Equi-width partitioning strategy works much better than other strategies such as K-means clustering for feature expressions with low to moderate rates of increase. In single query scenarios, aggressive read-ahead benefits both full aggregation as well as MOOLAP due to their sequential nature.

REFERENCES

- [1] C. Li, K. C.-C. Chang, and I. F. Ilyas, “Supporting ad-hoc ranking aggregates,” in *Proc. of SIGMOD*, 2002.
- [2] S. Antony, P. Wu, D. Agrawal, and A. E. Abbadi, “Towards multi-objective olap: Efficient skyline computation over ad-hoc aggregations,” in *UCSB Tech Report*, 2007.